

الغرض من هذا البرنامج هو تقديم نظام متكامل لإدارة الطلاب وأدائهم الأكاديمي، حيث يمكن من خلاله:

1. إضافة الطلاب وتسجيل معلوماتهم الأساسية مثل الاسم والرقم التعريفي.
2. إدخال الدرجات الأكاديمية لكل طالب بطريقة منظمة باستخدام قائمة انتظار (Queue) لضمان إدخال الدرجات بالترتيب.
3. حساب المعدل الدراسي لكل طالب وتحديد حالته الأكاديمية (ناجح أو راسب) باستخدام بنية البيانات struct لتخزين ملخص النتائج.
4. توليد تقارير أداء الطلاب وحفظها في ذاكرة مؤقتة باستخدام Stack لتسهيل استرجاع آخر تقرير تم إنشاؤه.
5. حفظ البيانات واسترجاعها من ملف خارجي لضمان استمرار المعلومات بين جلسات التشغيل.

C#

```
using System;
using System.Collections.Generic;
using System.IO;

class StudentSystem
{
    const int PASS_MARK = 50;
    const string DATA_FILE = "students_cs.txt";

    static Dictionary<string, Student> students = new
    Dictionary<string, Student>();

    // Queue: students waiting for marks
    static Queue<string> markQueue = new Queue<string>();

    // Stack: report history
    static Stack<string> reportStack = new Stack<string>();

    // Struct for result summary
    struct StudentResult
    {
        public double Average;
        public string Status;
    }

    class Student
    {
```

```
public string Name;
public List<int> Marks = new List<int>();
}

static void AddStudent()
{
    Console.Write("Student ID: ");
    string id = Console.ReadLine();
    Console.Write("Student Name: ");
    string name = Console.ReadLine();

    students[id] = new Student { Name = name };
    markQueue.Enqueue(id); // add to queue

    Console.WriteLine("Student added and queued for
marks.");
}

static void AddMarks()
{
    if (markQueue.Count == 0)
    {
        Console.WriteLine("No students in mark queue.");
        return;
    }

    string id = markQueue.Dequeue();

    Console.WriteLine($"Entering marks for:
{students[id].Name}");

    students[id].Marks.Clear();
    for (int i = 1; i <= 3; i++)
    {
        Console.Write($"Enter mark {i}: ");
        students[id].Marks.Add(int.Parse(Console.ReadLine()));
    }
}

static StudentResult CalculateResult(string id)
{
    double total = 0;
```

```

foreach (int m in students[id].Marks) total += m;

double avg = total / students[id].Marks.Count;

return new StudentResult
{
    Average = avg,
    Status = avg >= PASS_MARK ? "PASS" : "FAIL"
};
}

static void GenerateReport()
{
    Console.Write("Student ID: ");
    string id = Console.ReadLine();

    if (!students.ContainsKey(id) || students[id].Marks.Count
== 0)
    {
        Console.WriteLine("Invalid student or no marks found.");
        return;
    }

    StudentResult result = CalculateResult(id);

    string report =
        $"Name: {students[id].Name}\n" +
        $"Average: {result.Average}\n" +
        $"Status: {result.Status}";

    Console.WriteLine(report);

    reportStack.Push(report); // push to stack
}

static void ViewLastReport()
{
    if (reportStack.Count == 0)
    {
        Console.WriteLine("No reports available.");
        return;
    }
}

```

```
Console.WriteLine("\nLast Generated Report:");
Console.WriteLine(reportStack.Peek());
}

static void Save()
{
    using (StreamWriter sw = new StreamWriter(DATA_FILE))
    {
        foreach (var s in students)
        {
            sw.WriteLine($"{s.Key}|{s.Value.Name}|
{string.Join(",", s.Value.Marks)}");
        }
    }
}

static void Load()
{
    if (!File.Exists(DATA_FILE)) return;

    foreach (var line in File.ReadAllLines(DATA_FILE))
    {
        var parts = line.Split('|');
        students[parts[0]] = new Student
        {
            Name = parts[1],
            Marks = new
List<int>(Array.ConvertAll(parts[2].Split(','), int.Parse))
        };
    }
}

static void Main()
{
    Load();

    while (true)
    {
        Console.WriteLine(
            "\n1 Add Student" +
            "\n2 Add Marks (Queue)" +
```

```
"\n3 Generate Report" +  
"\n4 View Last Report (Stack)" +  
"\n5 Exit");
```

```
string c = Console.ReadLine();
```

```
if (c == "1") AddStudent();  
else if (c == "2") AddMarks();  
else if (c == "3") GenerateReport();  
else if (c == "4") ViewLastReport();  
else if (c == "5") { Save(); break; }  
}  
}  
}
```

Vb.net

```
Imports System
Imports System.Collections.Generic
Imports System.IO
```

```
Module StudentSystem
```

```
    Const PASS_MARK As Integer = 50
    Const DATA_FILE As String = "students_vb.txt"
```

```
    ' Dictionary to store students
    Dim students As New Dictionary(Of String, Student)()
```

```
    ' Queue for students waiting for marks
    Dim markQueue As New Queue(Of String)()
```

```
    ' Stack for report history
    Dim reportStack As New Stack(Of String)()
```

```
    ' Struct for result summary
    Structure StudentResult
        Public Average As Double
        Public Status As String
    End Structure
```

```
    Class Student
        Public Name As String
        Public Marks As New List(Of Integer)()
    End Class
```

```
    Sub AddStudent()
        Console.Write("Student ID: ")
        Dim id As String = Console.ReadLine()
        Console.Write("Student Name: ")
        Dim name As String = Console.ReadLine()

        students(id) = New Student With {.Name = name}
        markQueue.Enqueue(id) ' Add to queue

        Console.WriteLine("Student added and queued for
marks.")
    End Sub
```

```
End Sub

Sub AddMarks()
    If markQueue.Count = 0 Then
        Console.WriteLine("No students in mark queue.")
        Return
    End If

    Dim id As String = markQueue.Dequeue()
    Console.WriteLine($"Entering marks for: {students(id).Name}")

    students(id).Marks.Clear()
    For i As Integer = 1 To 3
        Console.Write($"Enter mark {i}: ")

students(id).Marks.Add(Integer.Parse(Console.ReadLine()))
    Next
End Sub

Function CalculateResult(id As String) As StudentResult
    Dim total As Double = 0
    For Each m As Integer In students(id).Marks
        total += m
    Next

    Dim avg As Double = total / students(id).Marks.Count
    Return New StudentResult With {
        .Average = avg,
        .Status = If(avg >= PASS_MARK, "PASS", "FAIL")
    }
End Function

Sub GenerateReport()
    Console.Write("Student ID: ")
    Dim id As String = Console.ReadLine()

    If Not students.ContainsKey(id) OrElse
students(id).Marks.Count = 0 Then
        Console.WriteLine("Invalid student or no marks found.")
        Return
    End If
End Sub
```

```
End If

Dim result As StudentResult = CalculateResult(id)
Dim report As String = $"Name: {students(id).Name}" &
vbCrLf &
    $"Average: {result.Average}" & vbCrLf &
    $"Status: {result.Status}"

Console.WriteLine(report)
reportStack.Push(report) ' Add to stack
End Sub

Sub ViewLastReport()
    If reportStack.Count = 0 Then
        Console.WriteLine("No reports available.")
        Return
    End If

    Console.WriteLine(vbCrLf & "Last Generated Report:")
    Console.WriteLine(reportStack.Peek())
End Sub

Sub Save()
    Using sw As New StreamWriter(DATA_FILE)
        For Each s In students
            sw.WriteLine($"{{s.Key}}|{{s.Value.Name}}|
{String.Join(",", s.Value.Marks}}")
        Next
    End Using
End Sub

Sub Load()
    If Not File.Exists(DATA_FILE) Then Return

    For Each line In File.ReadAllLines(DATA_FILE)
        Dim parts() As String = line.Split("|"c)
        students(parts(0)) = New Student With {
            .Name = parts(1),
            .Marks = New List(Of Integer)
(Array.ConvertAll(parts(2).Split(","c), Function(x)
Integer.Parse(x)))
    End For
End Sub
```

```
}  
Next  
End Sub  
  
Sub Main()  
Load()  
  
While True  
    Console.WriteLine(vbCrLf & "1 Add Student" &  
        vbCrLf & "2 Add Marks (Queue)" &  
        vbCrLf & "3 Generate Report" &  
        vbCrLf & "4 View Last Report (Stack)" &  
        vbCrLf & "5 Exit")  
  
    Dim c As String = Console.ReadLine()  
  
    Select Case c  
        Case "1"  
            AddStudent()  
        Case "2"  
            AddMarks()  
        Case "3"  
            GenerateReport()  
        Case "4"  
            ViewLastReport()  
        Case "5"  
            Save()  
            Exit While  
    End Select  
End While  
End Sub  
  
End Module
```

Java

```
import java.io.*;
import java.util.*;

public class StudentSystem {

    private static final int PASS_MARK = 50;
    private static final String DATA_FILE = "students_java.txt";

    // HashMap to store students
    private static Map<String, Student> students = new
    HashMap<>();

    // Queue for students waiting for marks
    private static Queue<String> markQueue = new
    LinkedList<>();

    // Stack for report history
    private static Stack<String> reportStack = new Stack<>();

    // Class to store result summary (struct equivalent)
    static class StudentResult {
        double average;
        String status;

        StudentResult(double average, String status) {
            this.average = average;
            this.status = status;
        }
    }

    static class Student {
        String name;
        List<Integer> marks = new ArrayList<>();
    }

    private static void addStudent(Scanner sc) {
        System.out.print("Student ID: ");
        String id = sc.nextLine();
        System.out.print("Student Name: ");
        String name = sc.nextLine();
    }
```

```

Student s = new Student();
s.name = name;
students.put(id, s);

markQueue.add(id);
System.out.println("Student added and queued for
marks.");
}

private static void addMarks(Scanner sc) {
    if (markQueue.isEmpty()) {
        System.out.println("No students in mark queue.");
        return;
    }

    String id = markQueue.poll();
    Student s = students.get(id);

    System.out.println("Entering marks for: " + s.name);
    s.marks.clear();
    for (int i = 1; i <= 3; i++) {
        System.out.print("Enter mark " + i + ": ");
        s.marks.add(Integer.parseInt(sc.nextLine()));
    }
}

private static StudentResult calculateResult(String id) {
    Student s = students.get(id);
    double total = 0;
    for (int mark : s.marks) {
        total += mark;
    }
    double avg = total / s.marks.size();
    String status = avg >= PASS_MARK ? "PASS" : "FAIL";
    return new StudentResult(avg, status);
}

private static void generateReport(Scanner sc) {
    System.out.print("Student ID: ");
    String id = sc.nextLine();

```

```

        if (!students.containsKey(id) ||
students.get(id).marks.isEmpty()) {
            System.out.println("Invalid student or no marks
found.");
            return;
        }

        StudentResult result = calculateResult(id);
        Student s = students.get(id);

        String report = "Name: " + s.name + "\n" +
            "Average: " + result.average + "\n" +
            "Status: " + result.status;

        System.out.println(report);
        reportStack.push(report);
    }

    private static void viewLastReport() {
        if (reportStack.isEmpty()) {
            System.out.println("No reports available.");
            return;
        }

        System.out.println("\nLast Generated Report:");
        System.out.println(reportStack.peek());
    }

    private static void save() {
        try (BufferedWriter bw = new BufferedWriter(new
FileWriter(DATA_FILE))) {
            for (Map.Entry<String, Student> entry :
students.entrySet()) {
                String id = entry.getKey();
                Student s = entry.getValue();
                String marks = String.join(",",
s.marks.stream().map(String::valueOf).toArray(String[]::new));
                bw.write(id + "|" + s.name + "|" + marks);
                bw.newLine();
            }
        } catch (IOException e) {

```

```

        System.out.println("Error saving data: " +
e.getMessage());
    }
}

private static void load() {
    File file = new File(DATA_FILE);
    if (!file.exists()) return;

    try (BufferedReader br = new BufferedReader(new
FileReader(file))) {
        String line;
        while ((line = br.readLine()) != null) {
            String[] parts = line.split("\\|");
            Student s = new Student();
            s.name = parts[1];
            if (!parts[2].isEmpty()) {
                for (String m : parts[2].split(",")) {
                    s.marks.add(Integer.parseInt(m));
                }
            }
            students.put(parts[0], s);
        }
    } catch (IOException e) {
        System.out.println("Error loading data: " +
e.getMessage());
    }
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    load();

    while (true) {
        System.out.println("\n1 Add Student" +
"\n2 Add Marks (Queue)" +
"\n3 Generate Report" +
"\n4 View Last Report (Stack)" +
"\n5 Exit");

        String choice = sc.nextLine();
    }
}

```

```
switch (choice) {  
    case "1": addStudent(sc); break;  
    case "2": addMarks(sc); break;  
    case "3": generateReport(sc); break;  
    case "4": viewLastReport(); break;  
    case "5": save(); return;  
}  
}  
}
```

Ruby

```
PASS_MARK = 50
DATA_FILE = "students_rb.txt"

# Dictionary to store students
$students = {}

# Queue for students waiting for marks
$mark_queue = []

# Stack for report history
$report_stack = []

# Struct for result summary
StudentResult = Struct.new(:average, :status)

class Student
  attr_accessor :name, :marks
  def initialize(name)
    @name = name
    @marks = []
  end
end

def add_student
  print "Student ID: "
  id = gets.chomp
  print "Student Name: "
  name = gets.chomp

  $students[id] = Student.new(name)
  $mark_queue << id

  puts "Student added and queued for marks."
end

def add_marks
  if $mark_queue.empty?
    puts "No students in mark queue."
    return
  end
```

```
id = $mark_queue.shift
student = $students[id]

puts "Entering marks for: #{student.name}"
student.marks.clear
3.times do |i|
  print "Enter mark #{i+1}: "
  student.marks << gets.chomp.to_i
end
end

def calculate_result(id)
  student = $students[id]
  total = student.marks.reduce(0, :+)
  avg = total.to_f / student.marks.size
  status = avg >= PASS_MARK ? "PASS" : "FAIL"
  StudentResult.new(avg, status)
end

def generate_report
  print "Student ID: "
  id = gets.chomp

  if !$students.key?(id) || $students[id].marks.empty?
    puts "Invalid student or no marks found."
    return
  end

  result = calculate_result(id)
  student = $students[id]
  report = "Name: #{student.name}\nAverage:
#{result.average}\nStatus: #{result.status}"
  puts report
  $report_stack.push(report)
end

def view_last_report
  if $report_stack.empty?
    puts "No reports available."
    return
  end
end
```

```

end

puts "\nLast Generated Report:"
puts $report_stack.last
end

def save
  File.open(DATA_FILE, "w") do |file|
    $students.each do |id, student|
      marks = student.marks.join(",")
      file.puts "#{id}|#{student.name}|#{marks}"
    end
  end
end

def load
  return unless File.exist?(DATA_FILE)

  File.readlines(DATA_FILE).each do |line|
    id, name, marks_str = line.chomp.split("|")
    student = Student.new(name)
    student.marks = marks_str.split(",").map(&:to_i) unless
marks_str.nil? || marks_str.empty?
    $students[id] = student
  end
end

# Main program loop
load
loop do
  puts "\n1 Add Student"
  puts "2 Add Marks (Queue)"
  puts "3 Generate Report"
  puts "4 View Last Report (Stack)"
  puts "5 Exit"
  print "Choice: "
  choice = gets.chomp

  case choice
  when "1"
    add_student

```

```
when "2"  
  add_marks  
when "3"  
  generate_report  
when "4"  
  view_last_report  
when "5"  
  save  
  break  
end  
end
```